Università degli Studi di Cagliari
Corso di Laurea in Ingegneria Elettronica

ELEMENTI DI INFORMATICA

http://agile.diee.unica.it

A.A. 2015/2016

Docente: **Michele Marchesi**

ALGORITMI E LINGUAGGI

Algoritmo: etimologia

- **Parola entrata in uso negli anni ‘50 per sostituire la parola *algorismo* che designava il processo di calcolare con i numeri arabi**

- **Etimologia**
  - **Creduta nel medioevo (*ma tuttora sospettata da molti studenti*): dal greco *algiros* (doloroso) + *arithmos* (numero)**
  - **Quella vera: dal nome dell’autore di un testo di algebra (825 d.C.) Abu Ja’far Mohammed ibn Mûsâ al-Khowârizmî**

Algoritmi: concetti informali

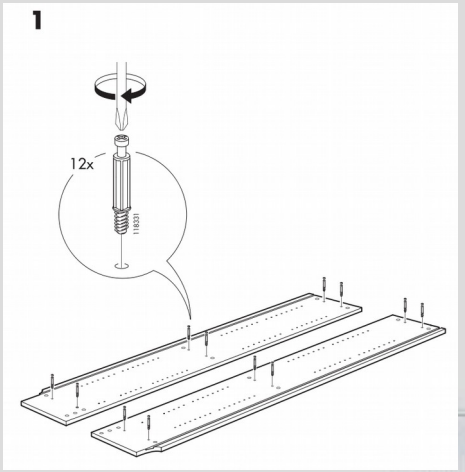
- Un algoritmo è una procedura computazionale **ben definita** che trasforma un insieme di dati di **ingresso** in un insieme di dati di **uscita**, al fine di risolvere un problema.

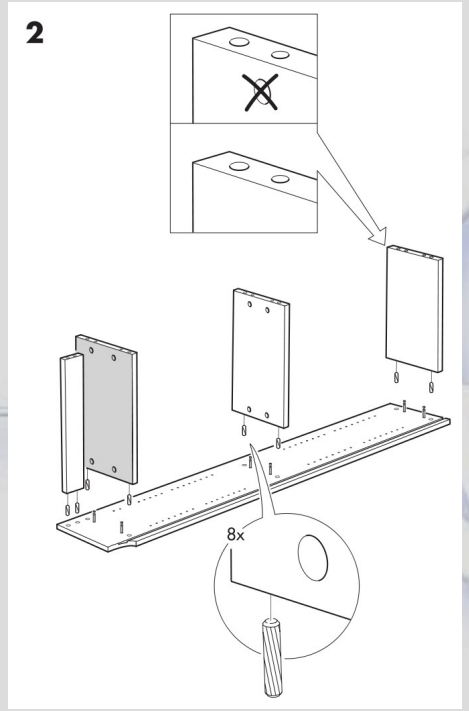
- Un algoritmo viene descritto da una **sequenza finita** di passi computazionali che hanno l'effetto di trasformare l'ingresso nell'uscita desiderata.

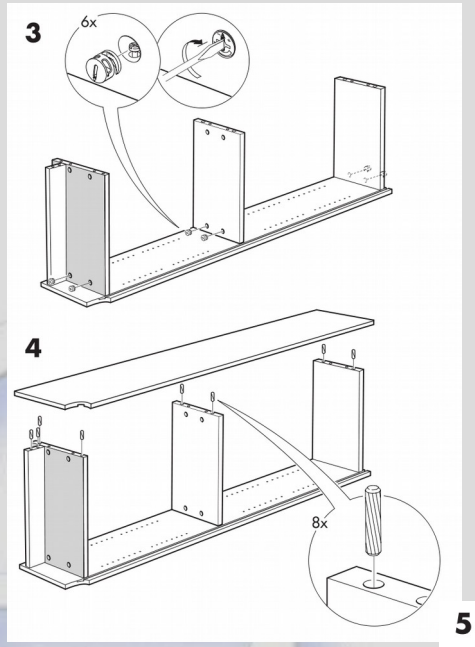
Algoritmo: concetti informali (cont.)

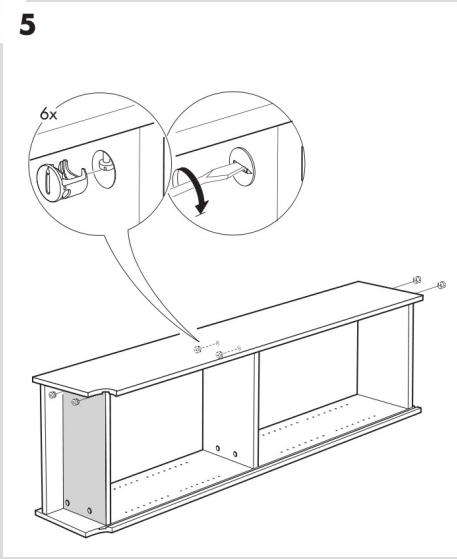
- **Esempi di "algoritmo" nella vita quotidiana:**
  - **Istruzioni di montaggio di un mobile**
  - **Calcolo del massimo comun divisore fra più numeri naturali**
  - **Prelevamento di denaro a un terminale Bancomat: inserire tessera magnetica, digitare codice, specificare importo da prelevare, ritirare la carta, prelevare il contante**
- **L'algoritmo deve essere *comprensibile* al suo esecutore**
  - **Se un libretto di istruzioni fosse disponibile solo in inglese non sarebbe compreso da chi conosce solo l'italiano.**

Montaggio di un mobile Ikea









Algoritmi e programmi

- In informatica, possiamo definire i calcolatori elettronici come esecutori di algoritmi.
- Gli algoritmi vengono descritti tramite **programmi**, cioè sequenze di istruzioni in un opportuno linguaggio comprensibile al calcolatore.
- Compito dell'esperto informatico:
  - **Produrre algoritmi**
  - **Codificarli in programmi**

Correttezza ed efficienza degli algoritmi

- Un'**istanza** di un problema è un particolare valore ammissibile assunto dai dati in ingresso
- Un algoritmo è **corretto** se perviene alla soluzione del compito senza difettare di alcun passo fondamentale
  - **termina** su **ogni istanza** del problema (la sequenza di passi computazionali è **finita**)
  - Produce l'uscita desiderata su **ogni istanza** del problema
- Un algoritmo è **efficiente** se perviene alla soluzione nel modo più veloce e/o usando la minor quantità di risorse

Esempio: utilizzo di una lavatrice

- Si vuole eseguire il programma “delicati”
  - Verifichiamo che la lavatrice sia collegata a una presa elettrica
  - Verifichiamo che il rubinetto di carico sia aperto
  - Carichiamo i capi da lavare
  - Carichiamo il detersivo e l'ammorbidente
  - Premiamo il tasto di programmazione ripetutamente sino a che non compare sul display il simbolo “delicati”
  - Premiamo il tasto “start”

Esempio: gestione di una biblioteca

- Una piccola biblioteca che contenga scaffali in cui sono disposti ii libri
  - Posizione dei libri invariabile negli scaffali
- Ad ogni libro è associata una scheda
  - Cognome e nome autori (nell'ordine in cui compaiono nel libro)
  - Titolo
  - Data di pubblicazione
  - Numero dello scaffale
  - Numero d'ordine della posizione nello scaffale

Esempio: gestione di una biblioteca

- Le schede sono conservate in ordine alfabetico rispetto al cognome del primo autore
- Semplice algoritmo per accedere ad un libro
  - Cercare la scheda nello schedario
  - Segnare su un foglietto scaffale e posizione del libro
  - Cercare lo scaffale indicato
  - Cercare il libro sullo scaffale. Se è presente, si compila il modulo di prestito con data e nome del richiedente

Algoritmo di 2 livello: ricerca in schedario

- Tecnica di ricerca nello schedario
  - Si esamina la prima scheda
  - Se il nome e il titolo coincidono con quelli cercati, la ricerca termina con successo, altrimenti si passa alla scheda successiva
  - Si continua con tutte le schede finché tutte le schede sono esaurite: se non si trova il libro, si deve cercare altrove
- Con questa tecnica di ricerca, nel caso migliore si esamina una scheda (la prima); nel caso peggiore si devono esaminare tutte le schede!

Algoritmo alternativa di ricerca

- Algoritmo più efficiente di ricerca
  - Si esamina la scheda *centrale* dello schedario
  - Se il nome e il titolo coincidono con quelli cercati, la ricerca termina con successo, altrimenti:
    - Se la scheda cercata segue in ordine alfabetico quella presa in esame, la ricerca continua nella seconda metà dello schedario;
    - Altrimenti la ricerca continua nella prima metà dello schedario
- Questo algoritmo si chiama ***ricerca binaria***

Linguaggi per la programmazione di algoritmi

- Nell'idea informale di algoritmo esiste il concetto implicito di **esecuzione di "istruzioni"**
- Queste istruzioni devono essere espresse in modo **preciso** e **non ambiguo**
- Se esprimiamo le istruzioni in linguaggio "naturale" l'ambiguità viene sostanzialmente rimossa dall'intelligenza umana e dalla conoscenza di un contesto informativo comune
  - Ma le istruzioni degli algoritmi vengono eseguiti da macchine che non hanno *buon senso!*

Linguaggi per programmare algoritmi

- Agli albori dell'informatica il linguaggio di programmazione coincideva con il linguaggio della macchina, cioè con l'insieme di comandi che la macchina era in grado di eseguire
- In pratica, si scrivevano direttamente le parole di memoria con sequenze di zero e uno interpretabili dal calcolatore
- Oppure si usava un linguaggio mnemonico in cui ogni riga (istruzione) corrisponde a una parola di memoria col programma: il linguaggio **Assembly**

Linguaggio Assembly

```
;*****************************************************************
; * FUNCTION: Calculates the square root of a 16-bit integer *
; * EXAMPLE : Number = FFFFh (65,535d), Root = FFh (255d)    *
; * ENTRY   : Number in File NUM:NUM+1                       *
; * EXIT    : Root in W. NUM:NUM+1; I:I+1 and COUNT altered  *
;*****************************************************************

; Local declarations
        cblock
         I:2, COUNT     ; Magic number hi:lo byte & loop count
        endc

        org    200h     ; Code to begin @ 200h in Program store
SQR_ROOT clrf  COUNT    ; Task 1: Zero loop count

        clrf   I        ; Task 2: Set magic number I to one
        clrf   I+1
        incf   I+1,f

; Task 3: DO
SQR_LOOP movf  I+1,w    ; Task 3(a): Number - I
        subwf  NUM+1,f  ; Subtract lo byte I from lo byte Num
        movf   I,w      ; Get high byte magic number
        btfss  STATUS,C ; Skip if No Borrow out
         addlw 1        ; Return borrow
        subwf  NUM,f    ; Subtract high bytes

; Task 3(b): IF underflow THEN exit
        btfss  STATUS,C ; IF No Borrow THEN continue
         goto  SQR_END  ; ELSE the process is complete

        incf   COUNT,f  ; Task 3(c): ELSE inc loop count

        movf   I+1,w    ; Task 3(d): Add 2 to the magic number
```

Linguaggi per programmare algoritmi

- Nella seconda metà degli anni ‘50 il linguaggio di programmazione si “alzò di livello”
  - Più adatto a codificare algoritmi
  - Più vicino al linguaggio naturale
- La *traduzione* del programma in linguaggio macchina è affidata alla macchina stessa
- Una singola riga (istruzione) del linguaggio corrisponde in genere a molte istruzioni macchina
- Si può anche opzionalmente tradurre il linguaggio di alto livello in istruzioni Assembly

Organizzazione *strutturata* di un calcolatore

- **Hardware** (circuiti digitali)
- **Microarchitettura** (esecuzione di istruzioni in linguaggio macchina)
- **Istruzioni in linguaggio macchina**
  - **Sistema operativo**
- **Linguaggio orientato ai problemi** (Fortran, Cobol, C, C++, Smalltalk, Java, C#, ecc.)
- A livello più alto si collocano linguaggi più vicini al linguaggio naturale usato per la descrizione di algoritmi.

Livelli di astrazione di un calcolatore

- Livello 0: *porte logiche*
  - Eseguono istruzioni logiche (AND, OR, ecc.) su segnali binari (0, 1)
- Livello 1: *Microarchitettura*
  - Insieme di porte logiche che eseguono operazioni su *insiemi di bit*
- Livello 2: ISA (*Instruction Set Architecture)*
  - E' l'insieme di istruzioni che possono essere eseguite dal μprocessore
- Livello 3: Sistema Operativo
  - Gestisce le risorse del calcolatore (μprocessore, memoria, periferiche, esecuzione di programmi utente)
- Livello 4: Linguaggi di programmazione di algoritmi – **Sono tradotti nel Livello 2**

Per poter eseguire un programma esistono due possibilità:

- Compilazione o traduzione
  - il programma è scritto in un linguaggio di alto livello (ad es. il C)
  - esso viene tradotto o compilato, generando l'eseguibile in linguaggio macchina (ad es. file .exe in Windows).
- Interpretazione
  - un programma chiamato interprete esegue ad una ad una le istruzioni del programma scritto nel linguaggio di alto livello (ad es. Java).
  - l'esecuzione è immediata (non c'è traduzione)

Compilazione

- Il programma in *codice sorgente* (ad es. in C) viene compilato dal compilatore
- Il risultato è del *codice oggetto*, tipicamente la traduzione del sorgente in assembly o codice macchina
- Assembly: linguaggio in corrispondenza 1:1 con le istruzioni macchina, ma leggibile
- Il codice oggetto è unito alle librerie dell'utente e di sistema (anch'esse in codice oggetto) dal *linker*
- Il codice risultante è l'eseguibile, in linguaggio macchina, che può essere caricato in memoria ed eseguito

Interpretazione

- L'interprete è un programma eseguibile, a sua volta scritto in un linguaggio di alto livello e tradotto
- L'interprete si chiama talora *macchina virtuale*
- Spesso l'interprete genera un formato intermedio più adatto all'interpretazione detto *bytecode*
- Poiché l'interprete gira in memoria, è possibile l'esecuzione immediata del codice, senza ciclo *compila-link-esegui*
- I programmi interpretati sono più lenti di quelli compilati, perché c'è un passo in più
- Gli interpreti moderni sono molto efficienti (compilazione *just in time*)

Linguaggi di *alto livello*

- Primo linguaggio di programmazione di alto livello: FORTRAN (FORmula TRANslator), per calcolo numerico
- Di poco posteriore il COBOL (COmmon Business Oriented Language), orientato alle applicazioni gestionali e all'elaborazione dati
- N.B. questi linguaggi, anche se datati, sono ancora molto diffusi ("legacy systems")

Linguaggi di *alto livello*

- Altri linguaggi basati su uno **studio dei principi della programmazione**:
  - Capostipite: ALGOL 60
  - Pascal: diffuso in passato nella didattica dell'informatica
  - C, il linguaggio attualmente più diffuso
  - ADA, usato dal Dip. della Difesa (DoD) degli USA
  - Linguaggi orientati agli oggetti: corrispondenza fra gli oggetti che caratterizzano una certa applicazione e la loro codifica. Ad es.: Smalltalk, C++, Java, C#, Phyton, Ruby, Javascript
  - Linguaggi per Web: Javascript, PHP

Ambienti di programmazione per linguaggi compilati

- Strumenti per facilitare la scrittura dei programmi in linguaggi di alto livello e verificarne la correttezza.
  - **Editor**: serve per scrivere il *programma sorgente*, cioè il **testo** che contiene le istruzioni nel linguaggio prescelto
  - **Compilatore**: Traduce un programma sorgente in *programma oggetto*, cioè in un programma in linguaggio macchina. Se vi sono errori di correttezza, viene avvisato il programmatore e il programma oggetto non viene generato.

Ambiente di programmazione

  - **Linker**: Collega insieme vari programmi oggetto che fanno parte di un unico programma suddiviso in *moduli* coordinati fra loro, generando il *programma eseguibile*.
  - **Debugger**: Consente di eseguire il programma passo passo verificando l'esecuzione delle istruzioni del programma sorgente e individuando eventuali errori

- Gli ambienti di programmazione forniscono in genere anche un insieme di *funzioni di libreria*, cioè di algoritmi comuni a molti programmi (es. lettura e scrittura di *file*, ordinamento di un vettore di numeri, ecc.)

Ambiente di programmazione per linguaggi interpretati

- Tre approcci possibili:
  - L'interprete fornisce un ambiente integrato in cui digitare ed eseguire i programmi
    - Es: Smalltalk, R
  - I programmi sono scritti con un Editor e poi eseguiti con l'interprete
  - Si utilizza un IDE (Integrated Development Environment – Ambiente di Sviluppo Integrato)
    - Es: Eclipse, Netbeans, Visual Studio
- L'esecuzione è più immediata (non c'è il passo di compilazione-linking) e il debugger più potente

Confronto compilatore/interprete

Compilatore:

- Opera su tutto il programma e crea un file eseguibile che gira direttamente nella CPU
  - Tale esecuzione è veloce
  - In caso di errore → crash
- Codifica più lenta e difficile
- Debug difficoltoso
- Protezione della proprietà intellettuale del codice

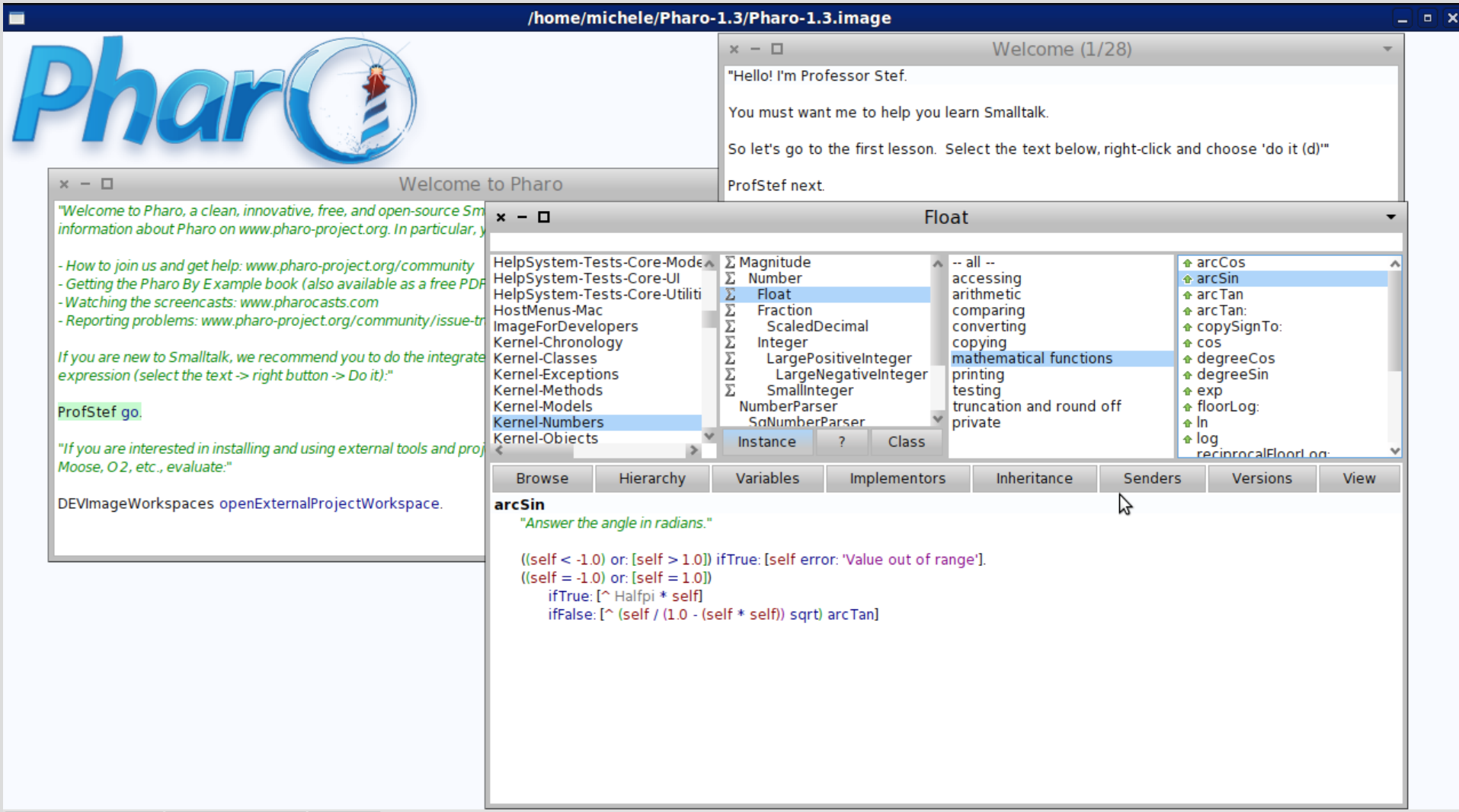
Interprete:

- Interpreta una riga di codice alla volta
  - L'interpretazione è lenta (in parte si ovvia usando *bytecode*)
  - In caso di errore l'interprete riprende il controllo
- Codifica più facile
- Debug più facile e potente
- Proprietà intellettuale del codice meno protetta

Linguaggio R

```
michele@michele-laptop:~$ R
R version 2.10.1 (2009-12-14)
Copyright (C) 2009 The R Foundation for Statistical Computing
R è un software libero ed è rilasciato SENZA ALCUNA GARANZIA.
Siamo ben lieti se potrai redistribuirlo, ma sotto certe
condizioni.
Scrivi 'license()' o 'licence()' per dettagli su come
distribuirlo.
> x <- 1:10
> x / 2
 [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> source("funz.R")
> x + a
 [1]  4  5  6  7  8  9 10 11 12 13
> q()
```

Linguaggio Smalltalk (Pharo)



Eclipse (Linguaggio Java)